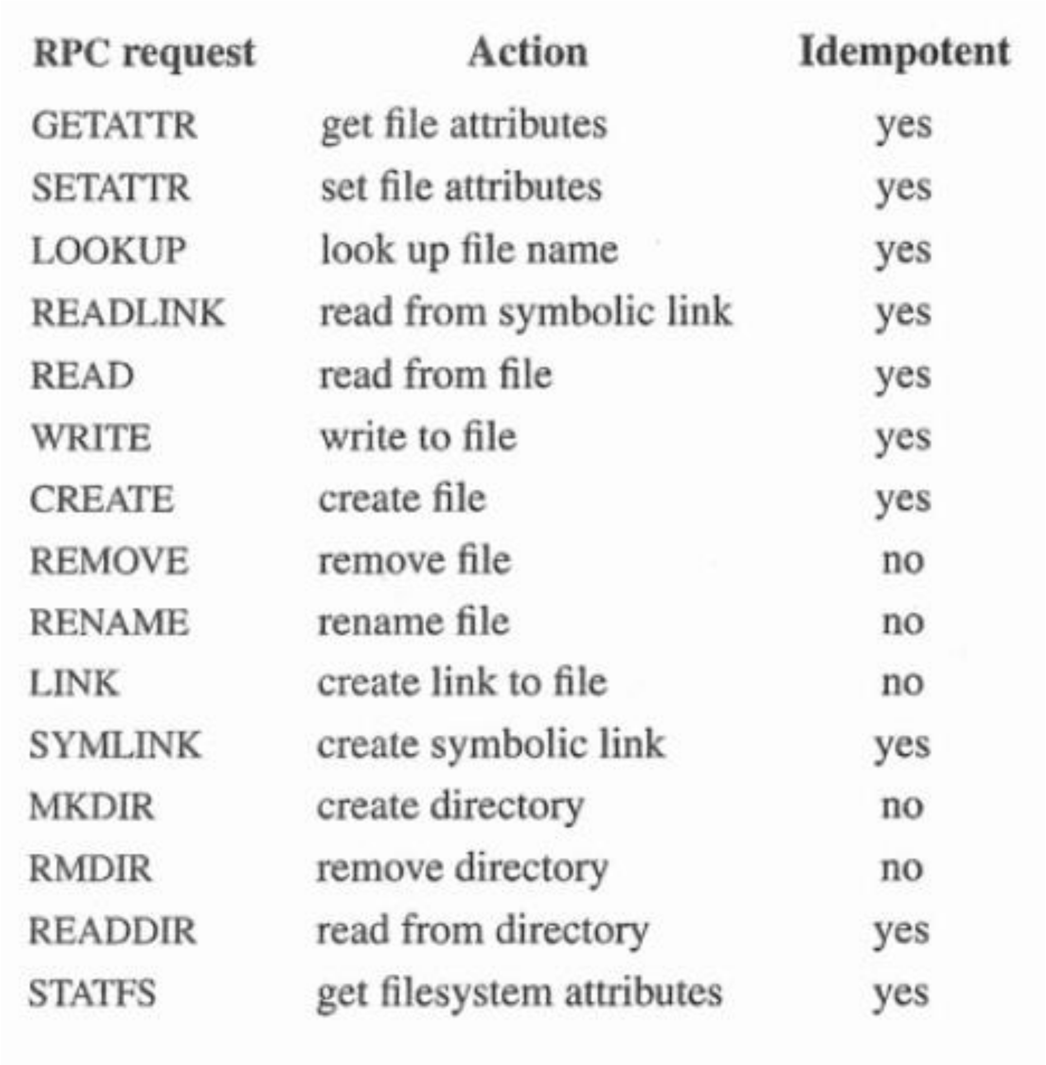
NFS Version 2 RPC requests



RPC request	Action	Idempotent
GETATTR	get file attributes	yes
SETATTR	set file attributes	yes
LOOKUP	look up file name	yes
READLINK	read from symbolic link	yes
READ	read from file	yes
WRITE	write to file	yes
CREATE	create file	yes
REMOVE	remove file	no
RENAME	rename file	no
LINK	create link to file	no
SYMLINK	create symbolic link	yes
MKDIR	create directory	no
RMDIR	remove directory	no
READDIR	read from directory	yes
STATFS	get filesystem attributes	yes

Stable-store requirement (NFS v2)

- All procedures in NFS v2 are synchronous
- When a procedure returns to the client, it assumes that the operation has completed and any data associated with the request is now on stable storage
- A WRITE may cause the server to update data blocks, indirect blocks, and attribute information (size, modify times)
- When the WRITE returns to the client, it can assume that the write is safe, even in case of a server crash

Statelessness

- Server has no information about clients, open files
  - Not entirely true in practice (e.g., retransmission cache)

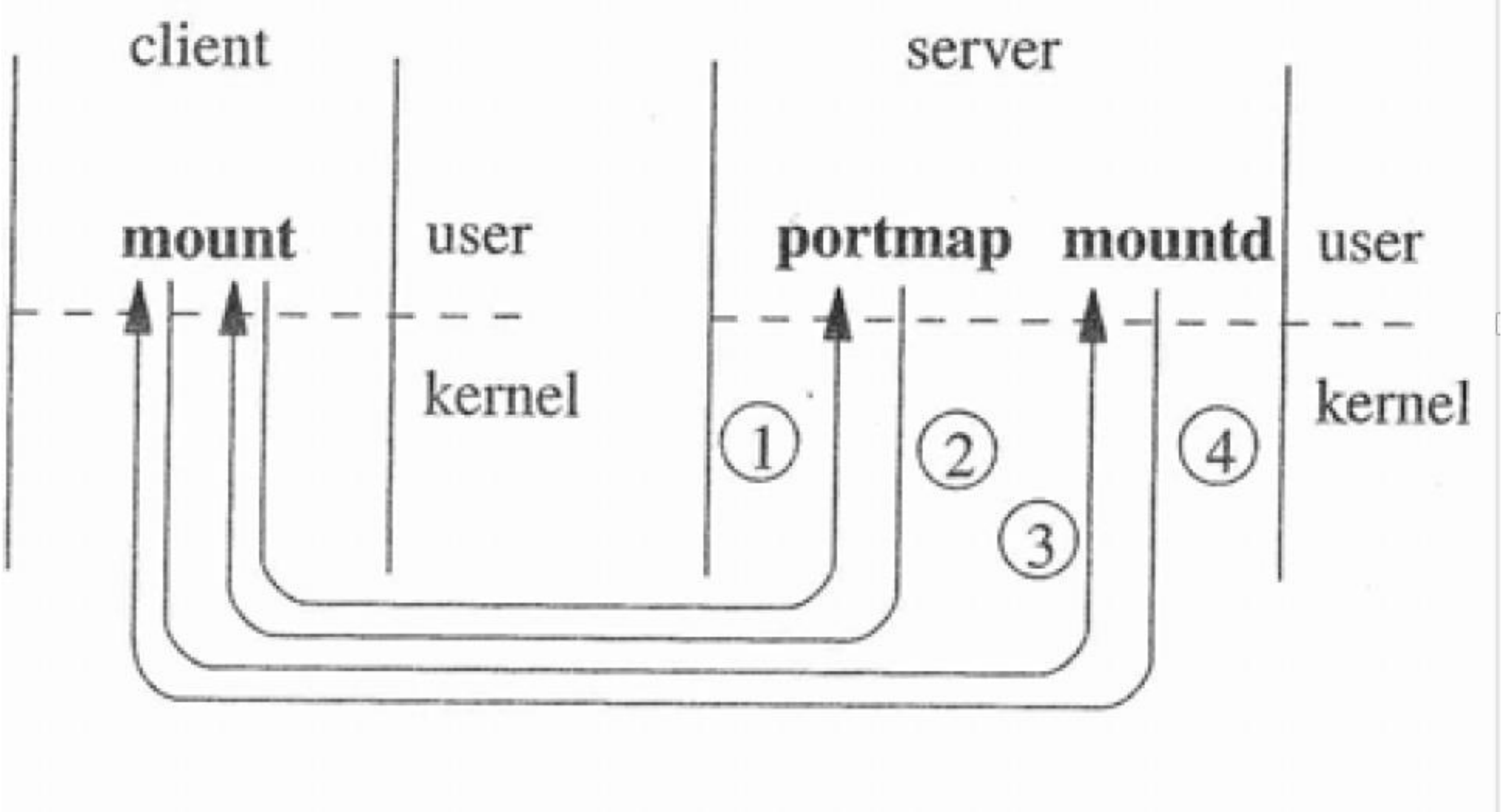
- Pros
  - Recovery

- Cons
  - Local file system semantics imply state
  - Performance (due to stable store requirement)
    - Addressed by NFS specs following v2

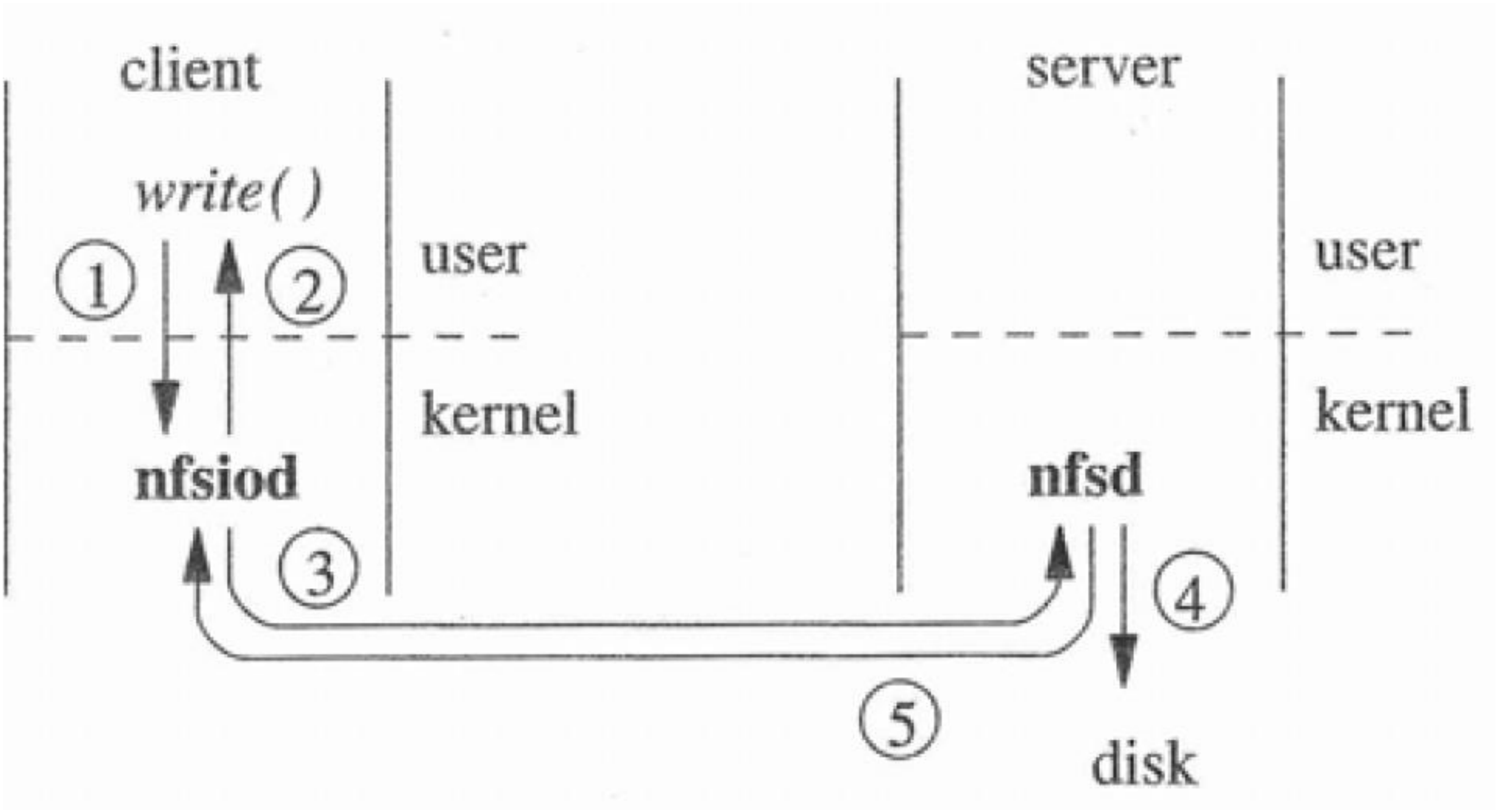
NFS Implementation

- Daemons
- Mounting a file system
- Performing I/O

Daemon interaction for mounting



Daemon interaction for I/O



Transport issues

- UDP implementation
  - RPC must fit in datagram (early versions restricted to 8KB)
  - Timeout, retransmission handled by RPC layer
  - Difficult to estimate RTT
  - RPC request broken down into IP fragments/Ethernet MTUs
- Or, run over TCP

Techniques for improving performance: Client caching and write buffering

- Delayed writes are allowed but cause problems
  - Other clients may see old versions of data
- Solution: close-to-open consistency
  - Clients flush on close(), so other clients will see the latest version on a later open()
- A write to server won’t be reflected on clients’ caches
  - No updates or invalidations (due to stateless server)
- Clients occasionally validate their caches
  - Send a GETATTR every 3 seconds and check the file's modification time and see if it has been updated

Existing solutions

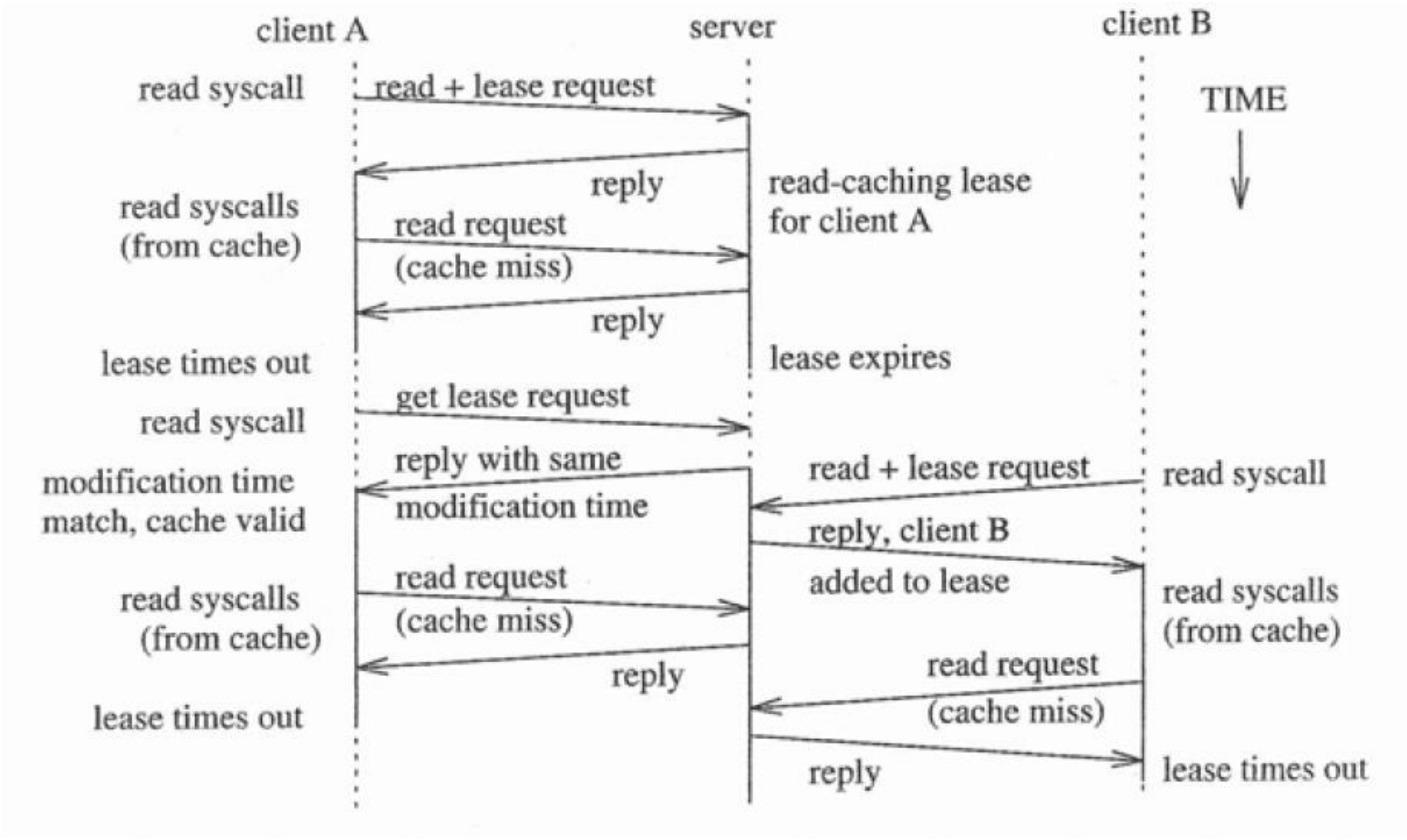
- NFS version 3
  - Leverage asynchronous writes
  - Writes back at 30′ intervals, flushes (commit) on close
  - Read consistency by checking with server on read (every 3″)
- Sprite [Nelson88]
  - File locks
  - Disable caching when detecting concurrent writes
- AFS [Howard88]
  - File locks
  - Callback-based invalidations

Leases: Fault-tolerant locks

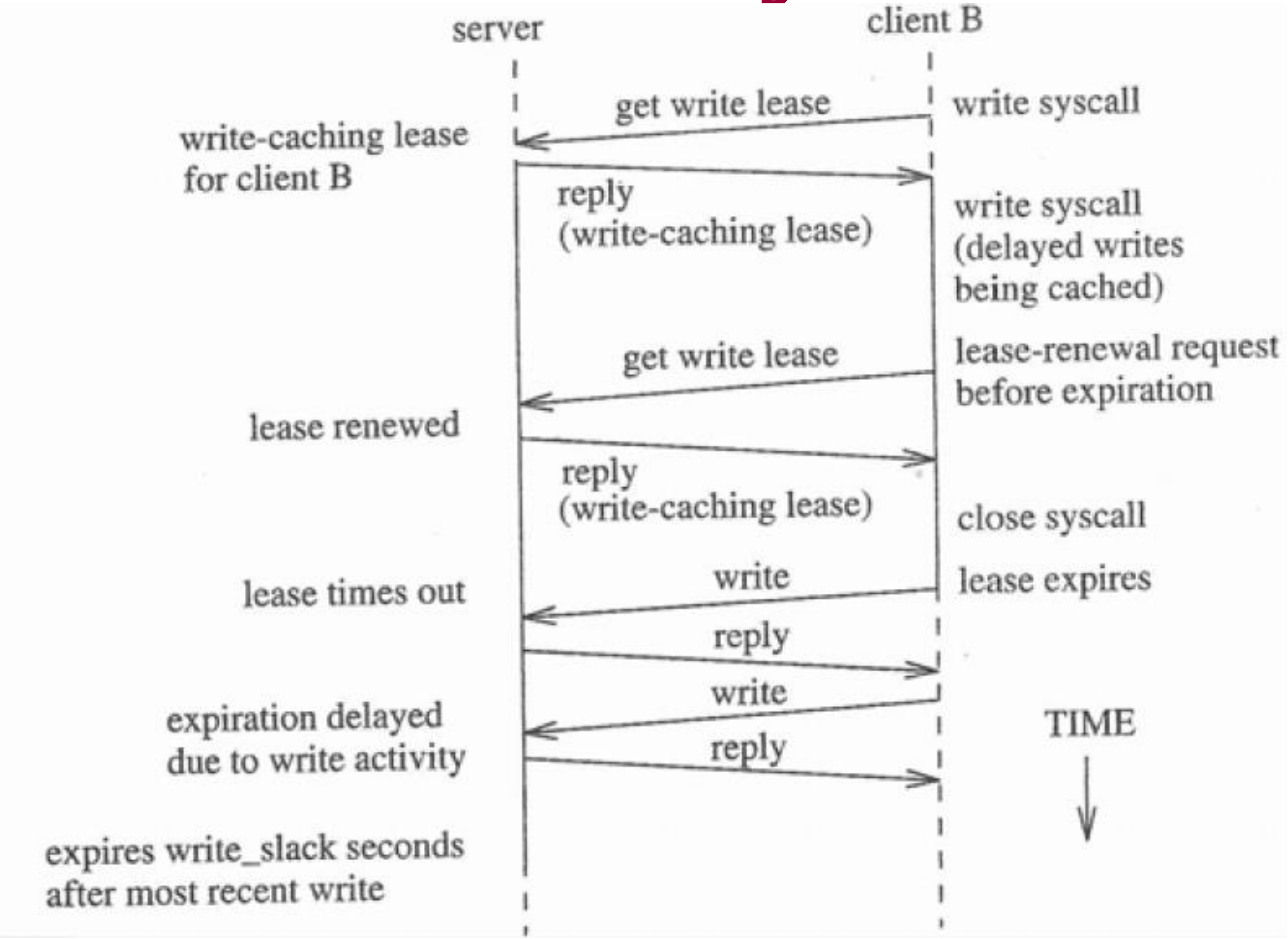
- Pros
  - Quick recovery without requiring hard state
  - Can tolerate partitions
  - Require moderate amount of soft state

- Important constants
  - `maximum_lease_term` (normally ~30”)
  - `clock_skew`
  - `write_slack`

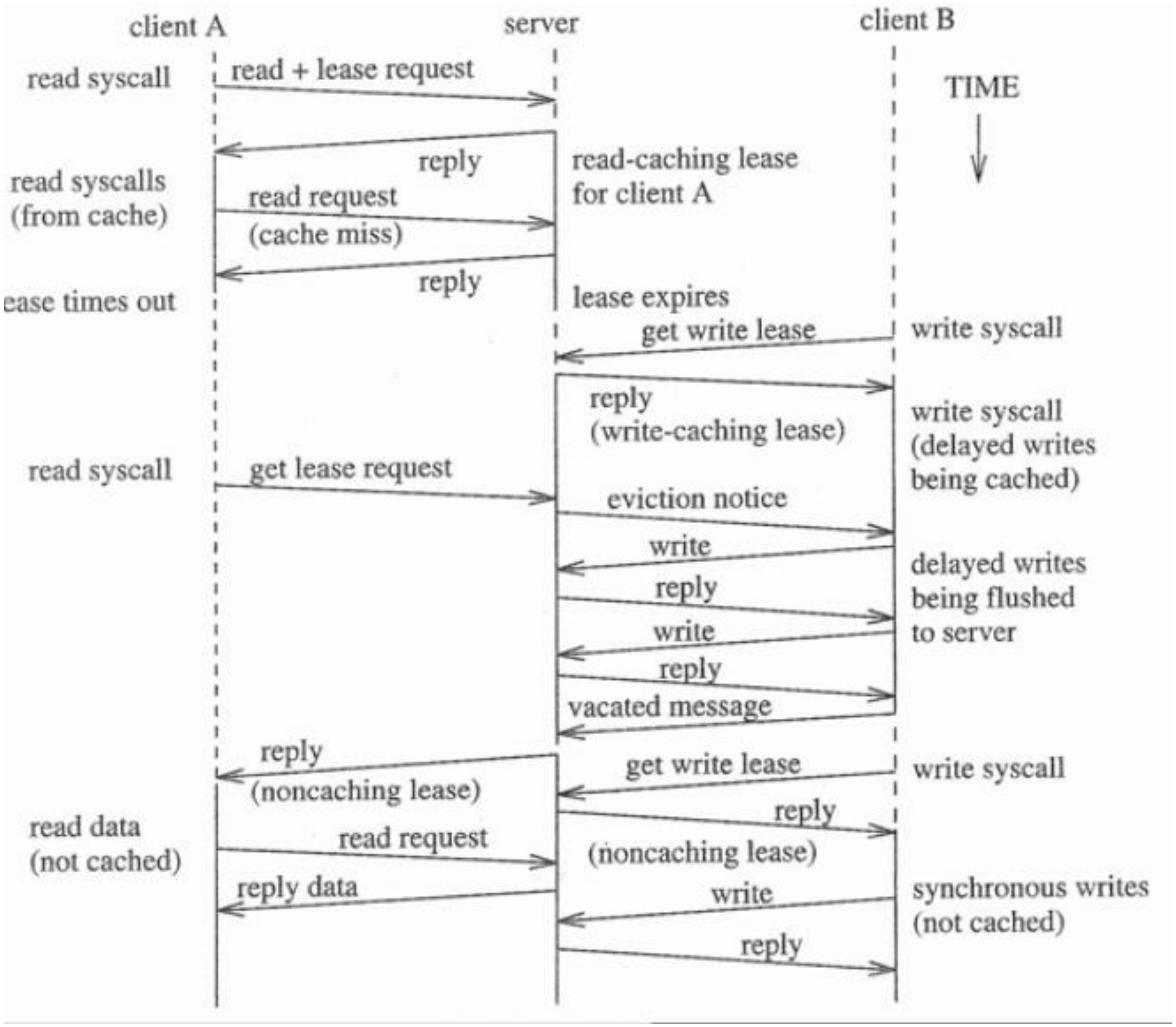
Read-caching leases



Write-caching lease



Write-sharing leases


client A
server
client B
TIME
read syscall
read + lease request
reply
read-caching lease
for client A
read syscalls
(from cache)
read request
(cache miss)
reply
ease times out
lease expires
get write lease
write syscall
reply
(write-caching lease)
write syscall
(delayed writes
being cached)
read syscall
get lease request
eviction notice
write
reply
write
reply
delayed writes
being flushed
to server
vacated message
reply
(noncaching lease)
get write lease
write syscall
reply
(noncaching lease)
read data
(not cached)
read request
reply data
write
reply
synchronous writes
(not cached)